华中科技大学

数据库系统原理实践报告

专　　业：	数据科学与大数据技术
班　　级：	大数据 2101 班
学　　号：	U202115652
姓　　名：	李嘉鹏
指导教师：	赵小松

分数	
教师签名	

2023 年 6 月 3 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

目　录

1 课程任务概述

本学期的《数据库系统原理实践》课程是《数据库系统原理》的配套实验课。实验课注重理论与实践相结合，其内容主要是对课内理论学习内容的具体化和拓展，要求我用 OpenGauss、Java 和 C++语言等实现数据库的一系列操作。具体来说，实验内容包括以下几个部分：

（1）数据库、表、索引、视图、完整性约束、存储过程与事务、函数、触发器、游标等数据对象的管理与编程，以及用户自定义函数的使用；

（2）数据查询（select）、数据插入（insert）、数据删除（delete）与数据修改（update）等数据处理相关任务；

（3）数据库的安全性控制、完整性控制，数据的备份与恢复机制、并发控制机制等系统内核的实验；

（4）数据库的设计与实现；

（5）数据库应用系统开发（Java 篇）；

（6）数据库的索引 B+树的实现。

课程主要依托于头歌实践教学平台，代码可以线上实时评测。实践课程链接的 url 为 https://www.educoder.net/classrooms/fpm4gqxz/shixun_homework。实验环境为 Linux 操作系统下的 OpenGauss2.1。在数据库应用开发环节，使用 Java 1.8。

2 任务实施过程与分析

本次实践课程在头歌平台进行，实践任务均在平台上提交代码，所有完成的任务、关卡均通过了自动测评。本次实践最终完成了任务书中的 2.1~2.12 的全部子任务以及 2.13、2.14 的部分子任务，总共在头歌平台上通过了 66 关，总分数为 133 分。下面将重点针对其中的 24 个任务阐述其完成过程中的具体工作。

2.1 数据库、表与完整性约束的定义(Create)

本小节主要是围绕数据库的创建以及表的各类约束（包括表的主码约束、外码约束、CHECK 约束、DEFAULT 约束、UNIQUE 约束）展开。

本任务已完成全部 6 个关卡。

2.1.1 创建数据库（第 1 关）

本关卡任务已完成，此处略过实验过程分析。（下文中仅列出要详细解释的关卡，不再单独说明简单的关卡）

2.1.2 CHECK 约束（第 4 关）

本关任务：创建表 products，并分别实现对品牌和价格的约束，主码约束不要显示命名。

实验过程：首先观察题目的要求，pid 为主码，brand 只能为'A'、'B'中的某一个，price 必须大于 0。据此即可分别创建一个主码约束和两个 CHECK 约束（分别命名为 CK_products_brand 和 CK_products_price）。CHECK 约束的语法是：CHECK 约束 ::= [CONSTRAINT [约束名]] CHECK (条件表达式)]。

需要注意的是，在代码开头处要先 drop 前一关的数据库，并创建一个新的数据库，从而实现对数据库的重置，否则可能会受到前面关卡的影响。代码如下。

```
drop database MyDb;
create database MyDb;
use MyDb;
create table products(
    pid char(10) primary key, name varchar(32),
    brand char(10) constraint CK_products_brand check(brand in
('A', 'B')),
    price int constraint CK_products_price check(price > 0));
```

2.1.3 DEFAULT 约束（第 5 关）

本关任务：创建表 hr，并分别为 id 创建主码约束、为 name 创建 NOT NULL 约束、为 mz 创建 DEFAULT 约束（缺省值为“汉族”）。

实验过程：按照题意写出各类约束即可。DEFAULT 约束的语法是：col_name data_type [DEFAULT {literal | (expr)}]。代码如下。

```
create table hr(
    id char(10) primary key, name varchar(32) not null,
    mz char(16) default '汉族');
```

2.2 表结构和完整性约束的修改(Alter)

本小节主要是围绕数据库表结构与完整性约束的修改操作展开。任务包括对数据库中表名/列名的修改、添加与删除字段、修改字段、添加列/表约束、删除与修改列/表约束等。

本任务已完成全部 4 个关卡。

2.2.1 添加、删除与修改约束（第 4 关）

本关任务：数据库中已经有 dept 和 staff 两个表，现在需要补充代码完成以下要求：(1) 为表 Staff 添加主码；(2) Dept.mgrStaffNo 是外码，对应的主码是 Staff.staffNo，添加这个外码，名字为 FK_Dept_mgrStaffNo；(3) Staff.dept 是外码，对应的主码是 Dept.deptNo，添加这个外码，名字为 FK_Staff_dept；(4) 为表 Staff 添加 check 约束，规则为 gender 的值只能为 F 或 M，约束名为 CK_Staff_gender；(5) 为表 Dept 添加 unique 约束，规则为 deptName 不允许重复。约束名为 UN_Dept_deptName。

实验过程：根据题目要求逐步写出各个约束的代码即可。

首先添加主码，语法是 ALTER TABLE 表名 ADD [CONSTRAINT [约束名]] PRIMARY KEY(列 1,列 2,...)；然后添加外码并给外码约束命名，语法是 ALTER TABLE 表名 ADD [CONSTRAINT [约束名]]；最后分别添加 CHECK 约束和 UNIQUE 约束，语法是 ALTER TABLE 表名 ADD [CONSTRAINT [约束名]] check(条件表达式)以及 alter table 表名 ADD [CONSTRAINT [约束名]] UNIQUE(列 1,...)。代码如下：

```
alter table Staff add primary key(staffNo);
```

```
alter table Dept add constraint FK_Dept_mgrStaffNo foreign
key(mgrStaffNo) references Staff(staffNo);
alter table Staff add constraint FK_Staff_dept foreign key(dept)
references Dept(deptNo);
alter table Staff add constraint CK_Staff_gender check(gender in
('F','M'));
alter table Dept add constraint UN_Dept_deptName unique(deptName);
```

2.3 数据查询(Select)

本小节主要覆盖了各类需求下的数据库查询任务。任务包括对数据库中数据的简单查询（例如直接用 select 语句配合简单的条件选择语句实现的查询）、对查询结果排序、多层嵌套查询与多条件查询、多表连接、外连接、对结果进行升序或降序排序、LIKE、NOT EXISTS 的用法、子查询、集函数与分组统计、top n 查询、数据消重、rank 函数的使用等等。

本任务已完成全部 25 个关卡。

2.3.1 商品收益的众数（第 6 关）

本关任务：查询资产表中所有资产记录里商品收益的众数（可能不止一个）和它出现的次数，出现次数命名为 presence。

实验过程：使用集函数 count(*)分别统计每个商品收益出现的次数，利用 all 求出全部次数的最大值，并找出出现次数大于等于这个最大值的商品收益，即可得到商品收益的众数。这里使用了嵌套查询，并利用 group by 子句实现了分组统计。代码如下。

```
select pro_income, count(*) as presence from property
group by pro_income
having count(*) >= all(
    select count(*) from property
    group by pro_income);
```

2.3.2 投资总收益前三名的客户（第 10 关）

本关任务：查询当前总的可用资产收益（被冻结的资产除外）前三名的客户的名称、身份证号及其总收益，按收益降序输出，总收益命名为 total_income。

实验过程：首先根据用户 id 对客户表 client 和财产表 property 进行自然连接，

然后按照用户名称和身份证号进行分组统计，利用 sum 函数计算每个用户总的可用资产收益，注意此处 pro_status 数据项的值需要为“可用”。最后利用 limit 函数输出前三名的客户即可。代码如下。

```
Select c_name, c_id_card, sum(pro_income) as total_income
from client, property
where c_id=pro_c_id and pro_status='可用'
group by c_name, c_id_card
order by total_income desc
limit 3;
```

2.3.3 客户理财、保险与基金投资总额（第 12 关）

本关任务：综合客户表(client)、资产表(property)、理财产品表(finances_product)、保险表(insurance)和基金表(fund)，列出客户的名称、身份证号以及投资总金额（每笔投资金额=商品数量*该产品每份金额）。注意投资总金额是客户购买的各类资产投资金额的总和，总金额命名为 total_amount。查询结果按总金额降序排序。

实验过程：这是一道比较复杂的查询题，主要是由于客户的投资总金额覆盖了理财产品、保险和基金三类，因此需要对这三个表进行连接，并按照用户名称和身份证号进行分组统计，取出每个表相应的商品数量与该产品单份投资金额，二者的乘积之和即为用户在该类别中的投资金额之和。最后，只需将三部分的和相加即可得到最终结果。实验中使用 union all 对三个表取可重复的并集，再使用 left outer join 与用户表做自然连接处理。这里需要特别小心，对于没有购买任何产品的客户，其 sum(pro_amount)的值可能为空值 NULL，因此需要使用 coalesce 函数对空值进行处理，强制将其赋为 0。代码如下。

```
select c_name, c_id_card,
    coalesce(sum(pro_amount), 0) as total_amount
from client
left outer join
(select pro_c_id, pro_quantity * p_amount as pro_amount
    from property, finances_product
    where pro_type=1 and p_id=pro_pif_id
union all
    select pro_c_id, pro_quantity * i_amount as pro_amount
    from property, insurance
```

```
    where pro_type=2 and i_id=pro_pif_id
union all
    select pro_c_id, pro_quantity *f_amount as pro_amount
    from property, fund
    where pro_type=3 and f_id=pro_pif_id) on c_id=pro_c_id
group by c_name,c_id_card
order by total_amount desc;
```

若没有使用 coalesce 函数，则结果会显示为空，且不会被正常降序排序，无法通过评测，如图 2.1 所示。

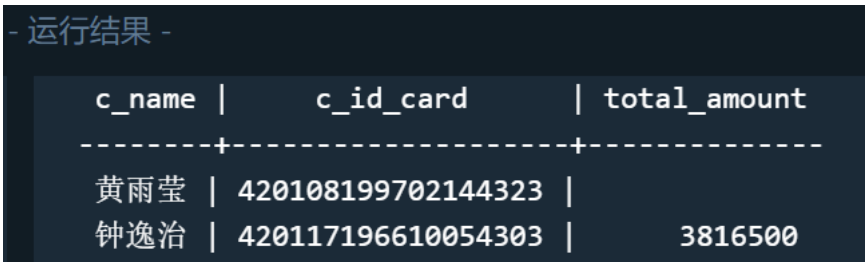

图 2.3.1：不对空值 NULL 赋值为 0 的结果

2.3.4 第 N 高问题（第 14 关）

本关任务：查询单份保险金额的第 4 高保险产品的编号和保险金额。排序时，相同的金额被视为同一排名。

实验过程：这道题可以先直接获取第 4 高的保险金额，然后再找出保险金额等于这个金额的所有保险产品。由于排名可以重叠，因此在选取金额时用 distinct 确保选取到正确的值，最后使用 limit 3,1 获得第 4 高的金额即可。代码如下。

```
select i_id, i_amount from insurance
where i_amount = (
    select distinct i_amount from insurance
    order by i_amount desc
    limit 3, 1);
```

2.3.5 基金收益两种方式排名（第 15 关）

本关任务：查询资产表中客户编号、客户基金投资总收益和其基金投资总收益的排名(从高到低)。总收益相同时名次并列。总收益命名为 total_revenue，名次命名为 rank。第一条 SQL 语句实现全局名次不连续的排名，第二条 SQL 语句实

现全局名次连续的排名。不管哪种方式排名，收益相同时客户编号小的都在前。

实验过程：以降序的基金投资总收益为关键字，直接使用 rank()和 dense_rank()函数即可，其中前者名次不连续，后者名次连续。

rank()函数的语法是：rank() over(partition by ...order by)，其中 partition by 用于给结果集分组，如果没有指定，那么视为把整个结果集作为一个分组。dense_rank()函数的语法与此类似。

实现名次不连续的代码如下：

```
select pro_c_id, coalesce(sum(pro_income), 0) as total_revenue,
    rank() over(order by total_revenue desc) as "rank"
from property where pro_type=3
group by pro_c_id;
```

实现名次连续的代码如下：

```
select pro_c_id, coalesce(sum(pro_income),0) as total_revenue,
    dense_rank() over(order by total_revenue desc) as "rank"
from property where pro_type=3
group by pro_c_id;
```

2.3.6 持有完全相同基金组合的客户（第 16 关）

本关任务：查询持有相同基金组合的客户对。如编号为 A 的客户持有的基金，编号为 B 的客户也持有；反过来，编号为 B 的客户持有的基金，编号为 A 的客户也持有，则(A,B)即为持有相同基金组合的二元组。最终只需要显示编号小者在前的那一对。

实验过程：这一题使用了两次 NOT EXISTS 条件判断语句。“两个客户持有的基金完全一致”这句话可以被转化成更清晰的意思，即不存在这样的基金是客户 A 持有而客户 B 不持有，也不存在这样的基金是客户 B 持有而客户 A 不持有，利用两层 NOT EXISTS 即可实现查询。需要注意两个客户必须至少购买一项基金，即两个没有购买任何基金的客户不计入统计。代码如下：

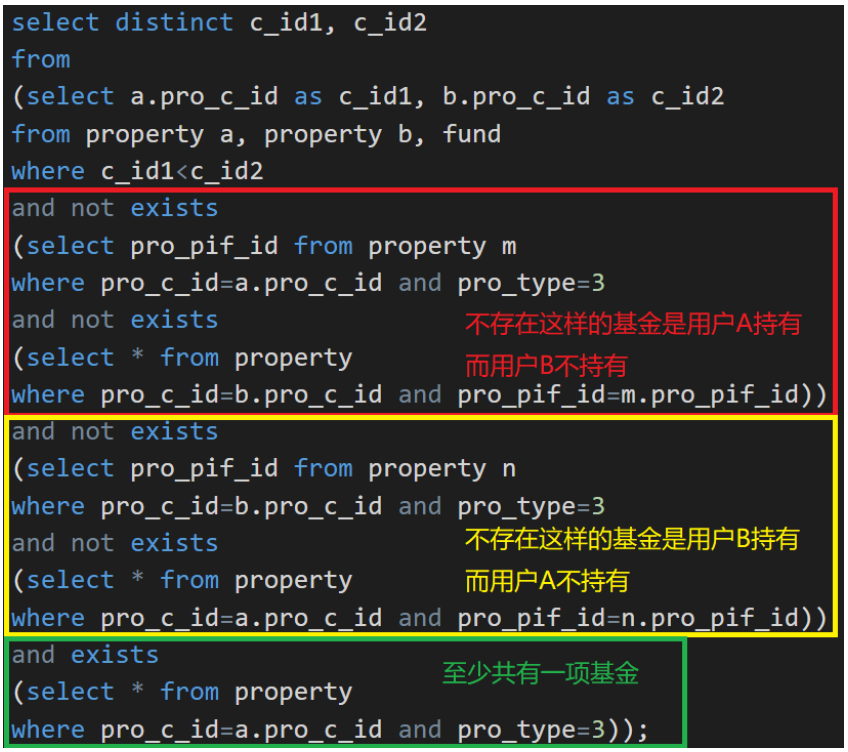

图 2.3.2：持有完全相同基金组合的客户代码解释

2.3.7 以日历表格式显示每日基金购买总金额（第 19 关）

本关任务：以日历表格式列出 2022 年 2 月每周每个交易日（周一到周五）基金购买的总金额，输出格式如表 2.3.1 所示。

表 2.3.1：日历表格式输出每周前五天基金购买的总金额

week_of_trading	Monday	Tuesday	Wednesday	Thursday	Friday
1	…	…	…	…	…
2	…	…	…	…	…
3	…	…	…	…	…
4	…	…	…	…	…

实验过程：对 2022 年 2 月 7 日开始的每一天进行处理。使用 date_part 函数获取每一个交易日的周次，由于 2 月的周次为 6-9，故需要将周次减去 5 作为 week_of_trading 值。同样地，利用 extract 函数从交易时间中提取处日期标记 daypos，daypos 从 1 到 5 分别对应周一到周五。这样处理后，对周一到周五的每一列，只会选取相应日期的基金购买总金额，否则相应位置设为空值 NULL。最终按照周次进行分组并排序。代码如下。

```sql
select wk as week_of_trading,
    sum(case when daypos = 1 then amount else null end) as monday,
    sum(case when daypos = 2 then amount else null end) as tuesday,
    sum(case when daypos = 3 then amount else null end) as
wendnesday,
    sum(case when daypos = 4 then amount else null end) as thursday,
    sum(case when daypos = 5 then amount else null end) as friday
from (
    select date_part('week', pro_purchase_time) -5 as wk,
    extract(DOW FROM cast(pro_purchase_time as TIMESTAMP)) as
daypos, sum(pro_quantity * f_amount) as amount
    from property
join fund on pro_pif_id = f_id
    where pro_purchase_time like '2022-02-%' and pro_type = 3
    group by pro_purchase_time)
group by wk
order by wk;
```

实现的日历表效果如图 2.3.3 所示。

运行结果

week_of_trading	monday	tuesday	wendnesday	thursday	friday
1	24000	30000	21000	55500	1190000
2	1780000	3215000	1830000	19000	45000
3	28000	41000	15000	22500	33000
4	15000				

(4 rows)

图 2.3.3：以日历表格式显示每日基金购买总金额

2.3.8 查询购买了所有畅销理财产品的客户（第 22 关）

本关任务：查询购买了所有畅销理财产品的客户编号(pro_c_id)。其中，畅销理财产品是指持有人数超过 2 的理财产品。结果按客户编号升序排列，且去除重复结果。

实验过程：这一关和前面 2.3.6 中“持有完全相同基金组合的客户”类似，也用到了二重 NOT EXISTS 条件判断语句。“一个用户购买了所有畅销理财产品”可以转化为“不存在这样一个畅销理财产品是这个用户没有购买的”。在第一层

NOT EXISTS 条件中，利用 having count(*)>2 筛选出所有畅销理财产品，并在第二层 NOT EXISTS 条件中逐个判断即可。代码如下。

```
select distinct pro_c_id from property a
where not exists
(select * from property b
    where b.pro_pif_id in
    (select pro_pif_id from property where pro_type=1
     group by pro_pif_id
     having count(*)>2)
and not exists
(select * from property c
     where c.pro_c_id=a.pro_c_id
     and c.pro_pif_id=b.pro_pif_id and c.pro_type=1))
order by pro_c_id;
```

2.3.9 查找相似的理财客户（第 25 关）

本关任务：查询每位客户(列名：pac)的相似度排名值小于 3 的相似客户(列名：pbc)列表，以及该客户和其相似客户共同持有的理财产品数(列名：common)和相似度排名值(列名：crank)。相似度的定义是：对于 A 客户，其购买的理财产品集合为{P}，另所有买过{P}中至少一款产品的其他客户集合为{B}，则{B}中每位用户购买的{P}中产品的数量为他与 A 客户的相似度值。将{B}中客户按照相似度值降序排列，得到 A 客户的相同相似度值。按照客户编号升序排列，取前两位客户即为 A 客户的相似理财客户列表。注意结果先按左边客户编号升序排列，同一个客户的相似客户则按客户相似度排名值顺序排列。

实验过程：按照题意逐步实现用户之间相似度的获取与排序。我创建了两个派生表，分别命名为 p1 和 p2，用于方便地两两枚举客户之间的组合。同时，利用 count(*)统计客户之间共同持有的理财产品数，并使用 rank() over 函数以共有的理财产品数为关键字进行降序排序，得到相似度排名值 crank。代码如下。

```
select * from
(select pro_c_id1 as pac, pro_c_id2 as pbc, count(*) as common,
rank() over(partition by pac order by common desc, pbc) as crank
   from
(select pro_c_id as pro_c_id1, pro_pif_id as pro_pif_id1
    from property where pro_type = 1)p1,
(select pro_c_id as pro_c_id2, pro_pif_id as pro_pif_id2
```

```
    from property where pro_type = 1)p2
  where pro_c_id1<>pro_c_id2 and pro_pif_id1=pro_pif_id2
  group by pro_c_id1, pro_c_id2)
where crank<3;
```

2.4 数据的插入、修改与删除(Insert,Update,Delete)

本小节主要是在不同场景下对数据库进行数据的插入、修改与删除操作。插入信息时，有完整和不完整两种情形；同时，还涉及到批量插入数据、修改数据属性、连接更新等。

本任务已完成全部 6 个关卡。

2.4.1 插入不完整的客户信息（第 2 关）

本关任务：向客户表 client 插入一条数据不全的记录。

实验过程：观察题目的输入数据，发现缺少了客户表的邮箱(c_mail)，因此在插入时用 NULL 赋空值。代码如下。

```
Insert into client
values(33,'蔡依婷
',NULL,'350972199204227621','18820762130','MKwEuc1sc6');
```

2.5 视图

本小节围绕视图的创建和基于视图的查询展开。

本任务已完成全部 2 个关卡。

2.5.1 插入不完整的客户信息（第 2 关）

本关任务：向客户表 client 插入一条数据不全的记录。

实验过程：利用视图的查询只需要将 from 语句后填写视图 view 的名称即可，代码较为简单，此处不再展示。

2.6 存储过程与事务

本小节围绕视图的创建和基于视图的查询展开。

本任务已完成全部 3 个关卡。

2.6.1 使用流程控制语句的存储过程（第 1 关）

本关任务：数据库中有表 fibonacci，用来储存斐波拉契数列的前 n 项，推导公式为：fib(n) = fib(n-1) + fib(n-2)。现在需要创建存储过程 sp_fibonacci(in m int)，向表 fibonacci 插入斐波拉契数列的前 m 项及其对应的斐波拉契数。fibonacci 表初始值为一张空表。

实验过程：首先创建存储过程，其语法格式是：create procedure 存储过程名(参数)。存储过程 sp_fibonacci(in m int)包含一个由 SQL 语句组成的主体，它是由以分号字符分隔的多个语句组成的复合语句。

具体来说，首先在 as 和 begin 之间的区域定义存储过程需要用到的变量，其中 a、b、temp 是递归过程中的三个数据变量，countnum 用于记录循环的轮数。对于 m 取值的不同，函数实现的路径不同。若 m 小于 3，则直接向 fibonacci 表中插入前两项（0,0）和（1,1）即可；否则，对更大的 m，则根据 fib(n) = fib(n-1) + fib(n-2)计算下一项 fibonacci 值即可。代码如下。

```
create procedure sp_fibonacci(in m int) as
    declare a int default 0;
    declare b int default 1;
    declare countunm int default 2;
    declare temp int default 0;
begin
    if m > 0 then
        insert into fibonacci values (0, 0); end if;
    if m > 1 then
        insert into fibonacci values (1, 1); end if;
    while m > countunm loop
        insert into fibonacci values (countunm, a + b);
        temp := a + b; a:= b; b:= temp; countunm:= countunm+1;
    end loop;
end;
```

2.6.2 使用游标的存储过程（第 2 关）

本关任务：实现基于游标的存储过程，其背景信息是：医院的某科室有科室主任 1 名(亦为医生)，医生若干(至少 2 名，不含主任)，护士若干(至少 4 人)，现在需要编写一存储过程自动安排某连续期间的大夜班(即每天 00:00-8:00)的值班表，排班规则为：每个夜班安排 1 名医生、2 名护士；值班顺序依工号顺序循环轮流安排(即排至最后 1 名后再从第 1 名接着排)；科室主任参与轮值夜班，但不

安排周末的夜班，当周末轮至科主任时，主任的夜班调至周一，由排在主任后面的医生依次递补值周末的夜班。本关卡中存储过程的名称为sp_night_shift_arrange，输入参数为 start_date 和 end_date，分别指排班的起始时间和结束时间。排班结果直接写入表 night_shift_schedule。

实验过程：这是一个比较复杂的存储过程，主要是要弄清楚游标在整个过程中是怎样发挥作用的。

首先，在 as 和 begin 之间的区域声明一些变量：cur_date 表示当前日期，leader_name 表示每晚轮值负责人的姓名，leader_adjustment 表示轮值负责人是否需要调整，d_name 表示医生的姓名，n_name1 和 n_name2 分别表示两名护士的姓名，d_type 表示人员类型（1 表示轮值负责人，2 表示医生，3 表示护士）。

接下来，使用游标 d_cursor 和 n_cursor 从 employee 表中获取所有类型不为 3 的人员和类型为 3 的人员。根据输入的起始日期和结束日期，依次处理每一天直到结束。在此过程中，需要判断当前选定的人员类型是否为 1（轮值负责人）且当前日期为周六或周日。这里可能出现三种情况：

①如果是，则将轮值负责人调整标志设置为 1，并获取下一人员插入值班表中，同时将当前日期加 1，并获取下一个护士的信息。

②如果当前日期为周一且轮值负责人调整标志为 1，则将调整标志设置为 0 并将轮值负责人插入值班表中，同时将当前日期加 1，并获取下两个护士的信息。

③如果当前日期不为周末且当前人员类型不为 1，则将当前人员插入值班表中，同时将当前日期加 1，并获取下一个医生和两个护士的信息。

如果游标 d_cursor 或 n_cursor 已经遍历完，则重新打开游标，继续获取人员信息，直到存储过程执行完毕。

本题最终通关的代码较长，此处使用图片展示，如图 2.6.1 所示。

```
4   create procedure sp_night_shift_arrange(in start_date date, in end_date date)
5   AS
6
7   declare cur_date date := start_date;
8   declare leader_name char(30);
9   declare leader_adjustment int := 0;
10  declare d_name char(30);
11  declare n_name1 char(30);
12  declare n_name2 char(30);
13  declare d_type int := 0;
14  declare cursor d_cursor for select e_name,e_type from employee where e_type != 3;
15  declare cursor n_cursor for select e_name from employee where e_type = 3;
16
17  begin
18      open d_cursor;
19      open n_cursor;
20      fetch d_cursor into d_name,d_type;
21      fetch n_cursor into n_name1;
22      fetch n_cursor into n_name2;
23      select e_name into leader_name from employee where e_type = 1;
24      while cur_date <= end_date loop
```

```
        if d_type = 1 and (extract(dow from cast(cur_date as timestamp)) = 0 or extract(dow from cast(cur_date as timestamp)) = 6) then
            leader_adjustment := 1;
            fetch d_cursor into d_name, d_type;
            if d_cursor % notfound then
                close d_cursor;
                open d_cursor;
                fetch d_cursor into d_name, d_type;
            end if;
            insert into night_shift_schedule values(cur_date, d_name, n_name1, n_name2);
            cur_date := cur_date + 1;
            fetch d_cursor into d_name, d_type;
            if d_cursor % notfound then
                close d_cursor;
                open d_cursor;
                fetch d_cursor into d_name, d_type;
            end if;
            fetch n_cursor into n_name1;
            if n_cursor % notfound then
                close n_cursor;
                open n_cursor;
                fetch n_cursor into n_name1;
            end if;
            fetch n_cursor into n_name2;
            if n_cursor % notfound then
                close n_cursor;
                open n_cursor;
                fetch n_cursor into n_name2;
            end if;
```

```
        elseif extract(dow from cast(cur_date as timestamp)) = 1 and leader_adjustment = 1 then
            leader_adjustment := 0;
            insert into night_shift_schedule values(cur_date, leader_name, n_name1, n_name2);
            cur_date := cur_date + 1;
            fetch n_cursor into n_name1;
            if n_cursor % notfound then
                close n_cursor;
                open n_cursor;
                fetch n_cursor into n_name1;
            end if;
            fetch n_cursor into n_name2;
            if n_cursor % notfound = true then
                close n_cursor;
                open n_cursor;
                fetch n_cursor into n_name2;
            end if;
        else
            insert into night_shift_schedule values(cur_date, d_name, n_name1, n_name2);
            cur_date := cur_date + 1;
            fetch d_cursor into d_name, d_type;
            if d_cursor % notfound then
                close d_cursor;
                open d_cursor;
                fetch d_cursor into d_name, d_type;
            end if;
            fetch n_cursor into n_name1;
            if n_cursor % notfound then
                close n_cursor;
                open n_cursor;
                fetch n_cursor into n_name1;
            end if;
            fetch n_cursor into n_name2;
            if n_cursor % notfound then
                close n_cursor;
                open n_cursor;
                fetch n_cursor into n_name2;
            end if;
        end if;
    end loop;

end;
```

图 2.6.1：使用游标的存储过程代码

2.6.3 使用事务的存储过程（第 3 关）

本关任务：在金融应用场景数据库中编写一个转账操作的存储过程 sp_transfer，实现从一个帐户向另一个帐户转账的功能。

实验过程：根据题意写出存储过程的判断条件，若合法则将返回值置为 1 并使用 COMMIT 提交事务，若不合法则将返回值置为 0 并使用 ROLLBACK 回滚对数据库的修改。具体条件是：①仅当转款人是转出卡的持有人时，才可转出；②仅当收款人是收款卡的持有人时，才可转入；③储蓄卡之间可相互转账；④允许储蓄卡向信用卡转账，称为信用卡还款(允许替他人还款)，还款可以超过信用卡余额，此时信用卡余额为负数；⑤信用卡不能向储蓄卡转账；⑥转账金额不能超过储蓄卡余额。

实现时只需要判断收款人和转款人的卡类别，并根据上述规则进行条件判断，若合法则使用 update 对双方的卡余额进行更新。代码如下。

```
create procedure sp_transfer(IN applicant_id int,
       IN source_card_id char(30), IN receiver_id int,
       IN dest_card_id char(30), IN amount numeric(10,2),
       OUT return_code int)
as
begin
    update bank_card set b_balance = b_balance - amount where b_type
= '储蓄卡' and b_number = source_card_id and b_c_id = applicant_id;
    update bank_card set b_balance = b_balance + amount where b_type
= '储蓄卡' and b_number = dest_card_id and b_c_id = receiver_id;
    update bank_card set b_balance = b_balance - amount where b_type
= '信用卡' and b_number = dest_card_id and b_c_id = receiver_id;
    if not exists (select * from bank_card where b_type = '储蓄
卡' and b_number = source_card_id and b_c_id = applicant_id and
b_balance >= 0) then
       return_code := 0; rollback;
    elseif not exists
    (select * from bank_card where b_number = dest_card_id and
b_c_id = receiver_id) then return_code := 0; rollback;
    Else return_code := 1; commit;
    end if; end;
```

2.7 触发器

本小节涉及了触发器的创建和使用。

本任务已完成全部 1 个关卡。

2.7.1 为投资表 property 实现业务约束规则-根据投资类别分别引用不同表的主码（第 1 关）

本关任务：为表 property(资产表)编写一个触发器，以实现以下完整性业务规则：①如果 pro_type = 1，则 pro_pif_id 只能引用 finances_product 表的 p_id；②如果 pro_type = 2，则 pro_pif_id 只能引用 insurance 表的 i_id；③如果 pro_type = 3，则 pro_pif_id 只能引用 fund 表的 f_id；④pro_type 不接受(1,2,3)以外的值。

实验过程：首先定义 msg 为出错信息（不超过 128 个字符），当数据不合法时用 raise exception 语句抛出异常，并设置出错信息：raise exception '%', msg。接下来根据题目条件构造出错语句格式即可实现触发器业务。代码如下。

```
CREATE OR REPLACE FUNCTION TRI_INSERT_FUNC() RETURNS TRIGGER AS
$$ DECLARE  --此处用 declare 语句声明你所需要的变量
   msg varchar(128);
BEGIN  --此处插入触发器业务
if new.pro_type <> 1 and new.pro_type <> 2 and new.pro_type <> 3
then msg := concat('type ',new.pro_type,' is illegal!');
    raise exception '%',msg; end if;
if new.pro_type = 1 and not exists(select * from finances_product
where finances_product.p_id = new.pro_pif_id) then
    msg := concat('finances product #',new.pro_pif_id,' not
found!'); raise exception '%',msg; end if;
if new.pro_type = 2 and not exists(select * from insurance where
insurance.i_id = new.pro_pif_id) then
    msg := concat('insurance #',new.pro_pif_id,' not found!');
    raise exception '%',msg; end if;
if new.pro_type = 3 and not exists(select * from fund where
fund.f_id = new.pro_pif_id) then
    msg := concat('fund #',new.pro_pif_id,' not found!');
    raise exception '%',msg; end if;
   --触发器业务结束    return new;--返回插入的新元组
END; $$ LANGUAGE PLPGSQL;
```

2.8 并发控制与事务的隔离级别

本小节包括并发控制中的不可重复读、幻读、主动加锁保证可重复读、可串行化等内容。

本任务已完成全部 4 个关卡。

2.8.1 不可重复读（第 1 关）

本关任务：现有两个并发事务 t1 和 t2，分别定义在 t1.sql 和 t2.sql 代码文件中，需要构造“不可重复读”现象。t2 是发生不可重复读的事务，t1 在 t2 的两次连续读之间修改了数据。

实验过程：首先要弄清楚“不可重复读”的概念。不可重复读是指一个事务读取到某数据后，另一个事务修改了该事务并未修改的数据，但当第一个事务再次读取该数据时发现两次读取的结果不一样。实验中只需将事务隔离级别设置为 read uncommitted，并根据任务给定的执行顺序，利用 pg_sleep()函数实现程序内部等待即可。具体来说，只需确保 t2 的第一次读取在 t1 读后和修改前，t1 在 t2 提交后读取。两个事务 t1 和 t2 的代码如下。

```
-- 事务 1:  -- 请设置适当的事务隔离级别
set session transaction isolation level read uncommitted;
-- 开启事务   start transaction;
-- 时刻 1 - 事务 1 读航班余票:  insert into result(t,tickets)
select 1 t, tickets from ticket where flight_no = 'CZ5525';
-- 添加等待代码，确保事务 2 的第一次读取在事务 1 修改前发生
select pg_sleep(2);
-- 时刻 3 - 事务 1 修改余票，并立即读取:
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';
insert into result(t,tickets)
select 1 t, tickets from ticket where flight_no = 'CZ5525';
commit;
-- 时刻 6 - 事务 1 在 t2 也提交后读取余票
-- 添加代码，确保事务 1 在事务 2 提交后读取
select pg_sleep(1); insert into result(t,tickets)
select 1 t, tickets from ticket where flight_no = 'CZ5525';
```

```
-- 事务 2
set session transaction isolation level read uncommitted;
start transaction;
-- 时刻 2 - 事务 2 在事务 1 读取余票之后也读取余票
-- 添加代码，确保事务 2 的第 1 次读发生在事务 1 读之后，修改之前
select pg_sleep(1); insert into result(t,tickets)
select 2 t, tickets from ticket where flight_no = 'CZ5525';
-- 时刻 4 - 事务 2 在事务 1 修改余票但未提交前再次读取余票，事务 2 的两次读取结果应该不同  -- 添加代码，确保事务 2 的读取时机
```

```
select pg_sleep(2); insert into result(t,tickets)
select 2 t, tickets from ticket where flight_no = 'CZ5525';
-- 事务 2 立即修改余票
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';
-- 时刻 5 - 事务 2 读取余票（自己修改但未交的结果）：
insert into result(t,tickets)
select  2 t, tickets from ticket where flight_no = 'CZ5525';
commit;
```

2.8.2 幻读（第 2 关）

本关任务：构造两个事务并发执行时的“幻读”现象。

实验过程：首先要弄清楚“幻读”的概念。幻读是指一个事务读取到某数据后，另一个事务作了 insert 或 delete 操作，导致第一个事务再次读取该数据时发现数据变多了或者变少了(记录数量不一致)。本题中在第 1 次查询后，事务 t2 插入了一条航班信息并提交，第 2 次查询的记录数增多，才能发生“幻读”。因此只需要在事务 1 中补充一句 select pg_sleep(2)即可。

2.9 备份+日志：介质故障与数据库恢复

本小节包括数据库的备份和恢复的具体操作。

本任务已完成全部 1 个关卡。

2.9.1 备份和恢复（第 1 关）

本关任务：给定数据库 residents，先运行 test1_1.sh 作备份，然后会 drop 数据 residents，最后运行 test1_2.sh 来恢复数据，并检查恢复是否成功。

实验过程：备份过程中，使用 gs_dump 工具对数据库的内容进行导出。gs_dump 的常见参数有：“-U”为连接数据库的用户名，“-W”指定用户连接的密码，“-f”将导出文件发送至指定目录文件，“-p”指定服务器所侦听的 TCP 端口或本地 Unix 域套接字后缀以确保连接，“-F”选择导出文件格式。据此即可写出备份的指令语句，同理也可以写出恢复的指令语句，代码如下。

```
gs_dump -U gaussdb -W 'Passwd123@123'  -p 5432 residents -f
residents_bak.sql -F t
gs_restore -U gaussdb -W 'Passwd123@123' residents_bak.sql -p 5432
-d residents
```

2.10 数据库设计与实现

本小节主要涉及了数据库设计与实现的相关任务，包括从概念模型到 OpenGauss 实现、从需求分析到逻辑模型等。

本任务已完成 2 个关卡。

2.10.1 从概念模型到 MySQL 实现（第 1 关）

本关任务：给定一个机票订票系统，系统需要考虑用户（user）、旅客（passenger）、机场（airport）、航空公司（airline）、民航飞机（airplane）、航班常规调度表（flightschedule）、航班表、机票等实体以及它们之间的联系。具体细节 E-R 图如图所示。

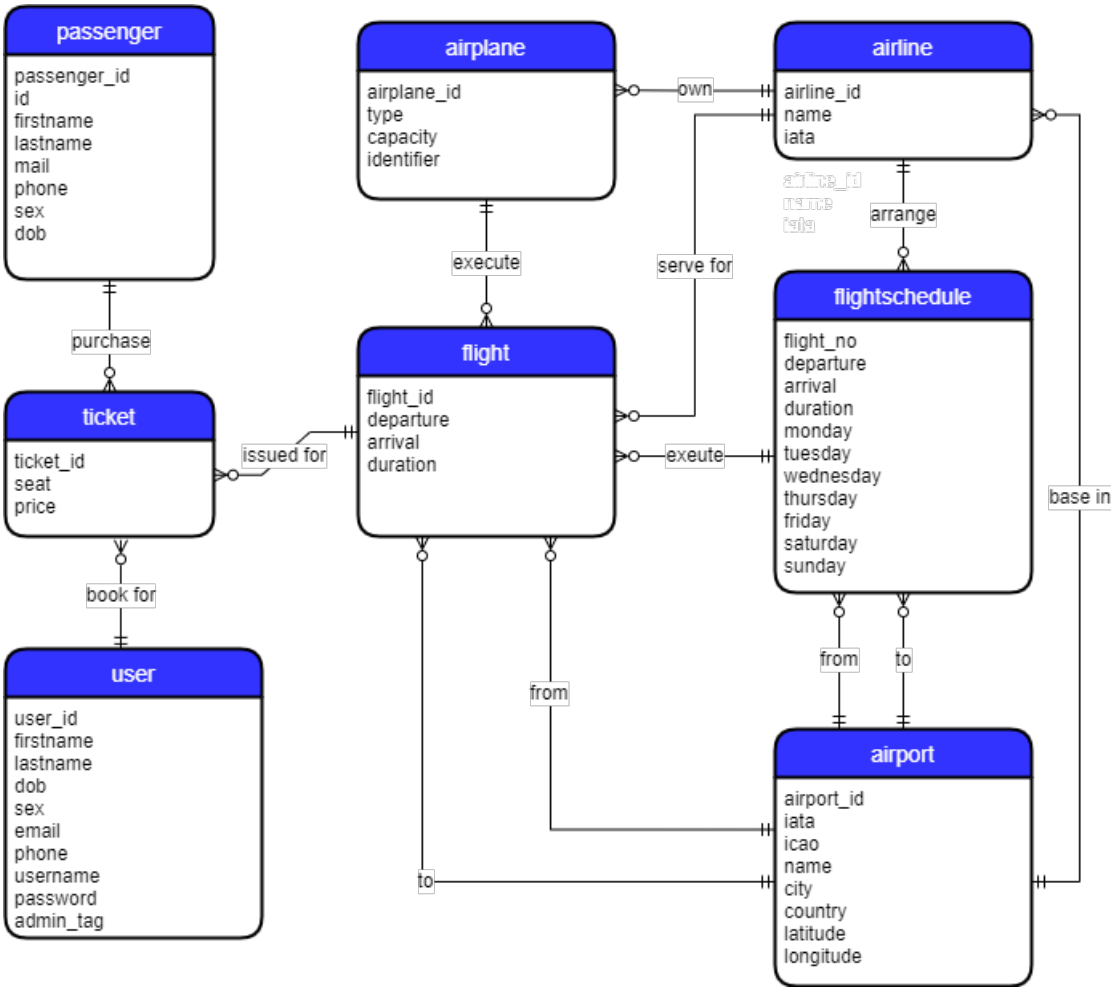

图 2.10.1：机票订票系统 E-R 图

现在需要根据上述信息和所给 E-R 图，在 OpenGauss 下实现 flight_booking 的语句，包括建表、创建主码、外码、索引、指定缺省、不能为空等约束。所有

索引采用 B TREE，所有约束的名字不作要求，所有的外码与主码同名，但有两处例外：计划航班和飞行航班都涉及出发机场和到达机场，外码与主码同名会导致同一表有两个同名列。故这两处外码例外处理：出发机场命名为 from，到达机场命名为 to。

实验过程：这是一个很庞大的系统，但实现时一步步按照题目给定的信息编写代码即可逐步完成。

第一步，要完成各个表的建立。建表时，要注意设置主码约束（PRIMARY KEY）、非空约束（NOT NULL）、不可重复约束（UNIQUE）、缺省值约束（DEFAULT）。

第二步，根据题目要求创建所有外码约束（foreign key），并自定义约束的名称。

第三步，依次创建全部的索引，并使用 B TREE 的形式，以方便查询等操作。

本关的代码较长，此处只列出部分，如图 2.10.2 所示。

```
# 请将你实现flight_booking数据库的语句写在下方：

drop table if exists "user" cascade;
create table "user"
(
    user_id int primary key,
    firstname varchar(50) not null,
    lastname varchar(50) not null,
    dob date not null,
    sex char(1) not null,
    email varchar(50) default '',
    phone varchar(30) default '',
    username varchar(20) not null,
    password char(32) not null,
    admin_tag tinyint not null default 0
);
create unique index username_unq on "user" using btree(username);

drop table if exists passenger cascade;
create table passenger
(
    passenger_id int primary key,
    id char(18) not null,
    firstname varchar(50) not null,
    lastname varchar(50) not null,
    mail varchar(50) default '',
    phone varchar(20) not null,
```

图 2.10.2：从概念模型搭配 MySQL 实现代码（节选）

2.10.2 从需求分析到逻辑模型（第 2 关）

本关任务：设计一个影院管理系统。影院对当前的放映厅和电影进行排片，顾客可购买任一排场的电影票，进入对应放映厅观看。系统中有以下实体集：电影、顾客、放映厅、排场、电影票。实体间的关系是：①顾客和电影票有一对多的购买关系；②电影票和排场有多对一的属于关系；③排场和电影有一对多的放映关系；④排场和放映厅有一对多的位于关系。

实验过程：根据题意绘制 E-R 图即可，并分析对应的关系模式。其中，E-R 图如图 2.10.3 所示。

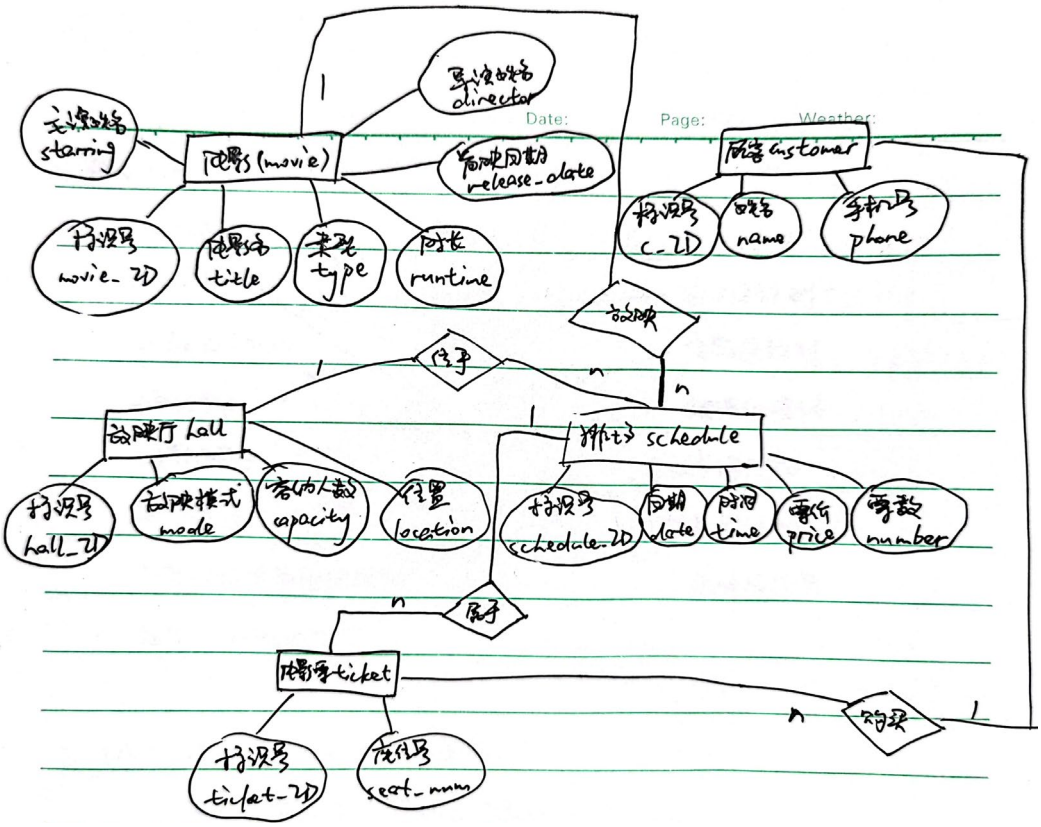

图 2.10.3：影院管理系统 E-R 图

图片链接和关系模式如下。

```
 请给出 ER 图文件存放的 URL：
https://gitee.com/GOODGAP/database/raw/master/QQ%E5%9B%BE%E7%89%8720230530124437.jpg    //ersolution.jpg

 以下给出关系模式：
电影(movie_ID, title, type, runtime, release_date, director, starring)，主码：movie_ID；
顾客(c_ID, name, phone)，主码：c_ID；
```

```
放映厅(hall_ID, mode, capacity, location)，主码：hall_ID;
排场(schedule_ID, date, time, price, number)，主码：schedule_ID;
电影票(ticket_ID, seat_num, movie_ID, phone)，主码：ticket_ID，外码：movie.movie_ID, customer.phone
```

2.10.3 制约因素分析与设计

在进行数据库的设计与实现时，总是会碰到大量的制约因素，这时需要结合实际情况设计符合要求的表结构与数据项，并构建出合理的数据库概念模型、逻辑模型并使用代码实现（例如 OpenGauss）。

具体来说，在第一关“机票订票系统”中，首先要设置好主码约束（PRIMARY KEY）、非空约束（NOT NULL）、不可重复约束（UNIQUE）、缺省值约束（DEFAULT）、外码约束（foreign key）等。这些是数据库表的最基本约束，是数据结构的基本属性。其次，在业务流程设计上，同样也有相关的约束，例如每个航班属于一家航空公司，航空公司可以很多航班，这体现了 1:n 联系；用户可以多次订票，旅客可以多次乘坐飞机，这体现了 m:n 联系。

又比如在第二关“影院管理系统”中，涉及了用户操作权限的制约。普通用户只能购买电影票，而管理员则可以查看后台数据，包括电影信息、顾客信息、排场信息等。

2.10.4 工程师责任及其分析

在本小节中，我深刻认识到了数据库应用所涉及的社会、健康、安全、法律以及文化等方面的重要性。我认为工程师应该有高度的责任心和使命感。

首先，数据库系统的设计和管理直接关系到社会信息化进程的发展，尤其是大数据时代下各行各业都需要运用数据库技术来存储、管理和分析数据，从而提高工作效率和服务质量。例如上面两关分别涉及到的机票订票系统和影院管理系统，都需要使用数据库来管理客户信息、购票结果等敏感数据，确保用户隐私得到保障。

其次，数据库系统的应用也对健康、安全和法律等因素产生了深远的影响。例如在食品安全方面，相关部门可以通过数据库技术收集和分析食品生产、流通、销售等环节的信息，及时发现问题并采取有效措施。同时，数据库系统也需要遵守国家相关法律法规，防止数据泄露和滥用，例如机票信息中包含的个人身份证

号就是非常敏感的数据。

综上所述，工程师在从事数据库应用开发和管理时，必须具备责任心，考虑各种因素对数据库系统的影响，并积极采取措施保障数据安全、稳定和可靠性。

2.11 数据库的索引 B+树实现

本小节主要是对 B+树基本数据结构的实现，以及在建立好的 B+树上实现 insert 和 remove 等操作。

本任务已完成 2 个关卡。

2.11.1 BPlusTreePage 的设计

本关任务：实现 BPlusTreePage 类，该类是 B+树叶结点类型和内部结点类型的父类，提供 B+树结点的基本功能。

实验过程：首先分析 BPlusTreePage 类。它作为 BPlusTreeInternalPage 与 BPlusTreeInternalPage 的父类，包含了 B+树结点的基本信息和功能，比如结点类型、包含元素存储最大值以及现存元素个数、父结点 id、当前结点 id 等。

接下来按照题意依次实现各个函数的功能。

①判断页类型是否为叶子结点或根结点，通过判断当前页的类型和父节点的 page_id 是否为 INVALID_PAGE_ID 实现。

②设置当前页的类型，通过 SetPageType 函数设置。

③获取或设置当前页存放的元素（键值对）个数，通过 GetSize、SetSize 和 IncreaseSize 函数实现。

④获取或设置当前页允许的最大元素个数，通过 GetMaxSize 和 SetMaxSize 函数实现。

⑤获取当前页允许的最少元素个数，通过 GetMinSize 函数实现。如果当前页是根节点，则根节点可能是内部节点或叶子节点。内部节点至少存在两个索引，叶子节点至少存在一条记录。对于非根节点，则至少要求是半满以上，如果是叶结点，至少需要满足[(max_size_-1)/2]个元素，如果是内部结点，至少需要满足第一个 key 不使用，(max_size_-1-1+1)/2+1 个元素。

⑥获取或设置当前页的父节点 page_id，通过 GetParentPageId 和 SetParentPageId 函数实现。

⑦获取或设置当前页的 page_id，通过 GetPageId 和 SetPageId 函数实现。

⑧设置 LSN（Log Sequence Number），用于恢复和日志管理。

这些操作函数是 B+树页的基本操作，支撑 B+树的插入、删除、查询等操作。最终各个函数的代码如下。

```
 * 函数功能：判断页类型是否为叶子结点
bool BPlusTreePage::IsLeafPage() const {
  return page_type_ == IndexPageType::LEAF_PAGE; }
 * 函数功能：判断页类型是否为根结点
bool BPlusTreePage::IsRootPage() const {
  return parent_page_id_ == INVALID_PAGE_ID; }
 * 函数功能：设置索引页类型
void BPlusTreePage::SetPageType(IndexPageType page_type)
{ page_type_ = page_type; }
 * 函数功能：get/set size (size：当前结点中存放的元素（键值对）个数)
int BPlusTreePage::GetSize() const { }
void BPlusTreePage::SetSize(int size) { size_ = size; return; }
void BPlusTreePage::IncreaseSize(int amount) { size_ += amount; }
 * 函数功能：get/set max size
int BPlusTreePage::GetMaxSize() const { return max_size_; }
void BPlusTreePage::SetMaxSize(int size) { max_size_ = size; }
 * 函数功能：获取当前结点允许的最少元素个数
int BPlusTreePage::GetMinSize() const {
  if (IsRootPage()) return IsLeafPage() ? 1 : 2;
  return (max_size_ + 1)/ 2; }
 *函数功能：get/set parent page id
page_id_t BPlusTreePage::GetParentPageId() const { return
parent_page_id_; }
void BPlusTreePage::SetParentPageId(page_id_t parent_page_id)
{ parent_page_id_ = parent_page_id; }
 *函数功能：get/set self page id
page_id_t BPlusTreePage::GetPageId() const { return page_id_; }
void BPlusTreePage::SetPageId(page_id_t page_id) { page_id_ =
page_id; }
 *函数功能：set lsn
void BPlusTreePage::SetLSN(lsn_t lsn) { lsn_ = lsn; }
```

3 课程总结

本次数据库系统原理实践课是理论与实践的结合、课内与课外的延伸。其总体任务包括对数据库的基本操作、对数据对象的管理与编程、数据库的安全性控制、完整性控制、并发控制机制、数据库的设计与实现，以及数据库的索引 B+树的实现。在实验中，我最终通过了 66 关，获得了 133 分的总分，对数据库多个维度的操作拥有了更深刻的认识和更熟练的技能。

通过对 OpenGauss 下 SQL 语法的学习，我第一次掌握了关系数据库的操作、维护和管理方法。我理解了如何创建数据库、表、索引、视图并定义其中数据项的类型，同时在面对一些实际要求时还需要创建各类约束，如主码约束、外码约束、非空约束等。面对实际情境下的数据库，我学会了对数据库进行更新、查询、备份与恢复，以满足具体的需要。我还学习了函数、触发器的定义与使用方法，探究了事务的并发控制机制，并对 B+树这一较复杂的数据结构略有涉足。同时，我还学会了从抽象的概念模型、逻辑模型中建立一个能真实运行的数据库。如果要用一句话来概括我的主要工作，那应该是在浅层学习了数据库操作的各类语法知识和常用操作，并在深层了解了数据库的内核处理思想。

本次数据库实验课给我带来了很丰富的收获。具体来说，我的心得体会有以下几点。在做数据查询时，我发现同一个查询需求可能对应了很多种不同的写法，也就是查询的路径多样化，十分灵活。通过创建索引和视图，可以使查询更方便。存储过程可以用来完成很多功能，例如实现日程的自动化安排。其中，利用游标的存储过程实现起来非常复杂，但最后还是在不断改进中成功通过了评测，让我学会了游标可以对一组数据进行逐行处理，增加了灵活性和精确性。在利用 E-R 图编写数据库代码时，我再次体会到了概念模型和实际需求分析的紧密联系。

本学期的数据库实验课程就到此结束了，由衷地感谢赵老师和潘老师在课上对我的耐心答疑与指导，同时也要感谢无数个夜晚坚持调 bug 的自己！